# METADAMAGENET: USING DEEP LEARNING TO IDENTIFY AND CLASSIFY DAMAGE IN AERIAL IMAGERY

**Nima Afshar**
Computer Science Bachelor's
Amirkabir University Of Technology
Tehran, Iran
nima.afsharh@gmail.com

**Amin Gheibi**
Assistant Professor
Amirkabir University Of Technology
Tehran, Iran
amin.gheibi@aut.ac.ir

**ABSTRACT**

This work is Nima Afshar's bachelor's thesis at AmirKabir University of Technology under the supervision of Dr.Amin Gheibi. In this work, we tackle the problem of identifying and classifying damage caused by natural disasters in aerial imagery on the xbd dataset [1]. The localization task is about finding buildings in pre-disaster images, and the classification task is about finding the damage level on buildings. We take Xview2 challenges first place solution [2] as the baseline and try to enhance its method, first, by changing the model structure to come up with a more accurate and more efficient one. Second, in attempting to use meta-learning to detect damage based on different features of different disasters (hurricanes, floods, etc.). Our experiments show that by applying some minor changes to the model structure, a model with fewer parameters can give us even more promising results compared to the models proposed in previous solutions. We used the mentioned repository as a baseline and refactored its code. Thus, this project covers models and experiments of the mentioned repo and contributes more to the same problem of damage assessment in aerial imagery.

***K*eywords** Semantic Segmentation · Damage Assessment · Vision Transformers · Meta Segmentation

## 1 Introduction

Detecting damaged areas in natural disasters is the first step in the damage assessment process. As soon as the damaged areas are detected, first responders can be sent to them to start their operation. Usually, the number of first responders and equipment available for rescue operations is limited. Thus, prioritizing damaged areas in terms of risk can lead to more efficient damage assessment. First responders and equipment can be sent to where a natural disaster has caused more damage and people or valuable assets are at higher risk. As a result, more lives and assets can be saved with the correct identification and prioritization of damage done to different areas by natural disasters. So, it is essential to reduce the damage identification time. There have been many successful usages of computer vision and deep learning for this task since having an automated model that detects damaged areas and their damage level based on easy-to-scrape data like aerial imagery can be a very valuable asset to each damage assessment team or organization. In this work, we build on previous experiences regarding identifying and classifying damaged areas based on their damage levels as a semantic segmentation task.

## 2 Dataset

We are using the xview2 [3] challenge dataset, namely Xbd [1], as the dataset for our project. This dataset contains pairs of pre and post-disaster satellite images from 19 natural disasters worldwide, including fires, hurricanes, floods, and earthquakes. Each sample in the dataset consists of a pre-disaster image with its building annotations and a post-disaster image with the same building annotations. However, in the post-disaster building annotations, each building has a damage level of the following: *undamaged*, *damaged*, *major damage*, *destroyed*, and *unclassified*. The dataset consists of *train*, *tier3*, *test*, and *hold* subsets. Each subset has an *images* folder containing pre and post-disaster images stored as $1024 \times 1024$ *PNG*s and a folder named *labels* containing building annotations and damage labels in *JSON* format.

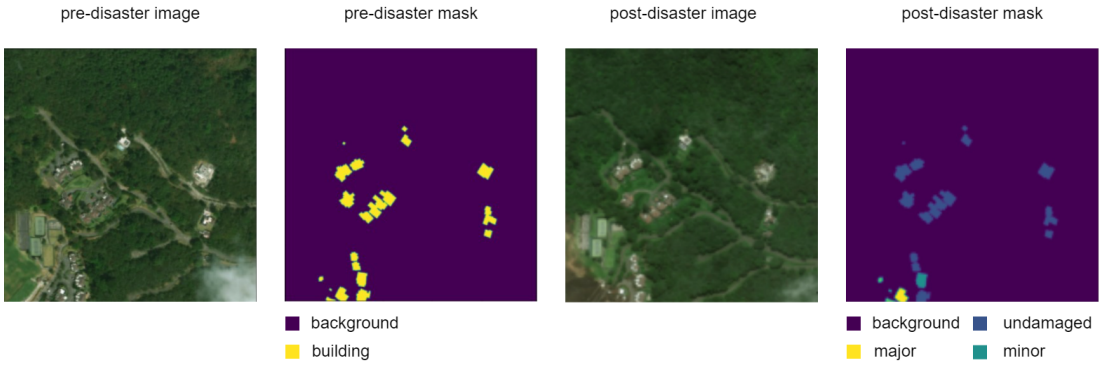


Figure 1: Aerial Imagery before and after the disaster with their corresponding masks

Some of the post-imagery is slightly shifted from their corresponding pre-disaster image. Also, the dataset has different ground sample distances. We used the *train* and *tier3* subsets for training, the *test* subset for validation, and the *hold* subset for testing. The dataset is highly unbalanced in multiple aspects. The buildings with the *undamaged* label are far more than buildings with other damage types. The number of images varies a lot between different disasters; the same is true for the number of building annotations in each disaster.

## 3 Problem Definition

We can convert these building annotations (polygons) to a binary mask. We can also convert the damage levels to values 1-4 and use them as the value for all the pixels in their corresponding building, forming a semantic segmentation mask. Thus, we define the building localization task as predicting each pixel's value being zero or non-zero. We also define the damage classification task as predicting the exact value of pixels within each building. We consider the label of an *unclassified* building as *undamaged*, as it is the most common label by far in the dataset.

## 4 Data Augmentations

Data Augmentation techniques help generate new valid samples from the dataset. Hence, they provide us with more data, help the model train faster, and prevent overfitting. Data Augmentation is vastly used in training computer vision tasks, from image classification to instance segmentation. In most cases, data augmentation is done randomly. This randomness means that the augmentation is not done on some of the original samples and it has some random parameters. Most libraries used for augmentation, like Open-CV [4], do not support image-batch transforms and only perform transforms on the CPU. Kornia [5] [6] is an open-source differentiable computer vision library for *PyTorch* [7]; it supports image-batch transforms and performs these transforms on GPU. We used Kornia and added some parts to it to suit our project requirements.

We created a version of each image transformation in order for it to support our needs. Its input is multiple batches of images, and each batch has a name. For example, an input contains a batch of images and a batch of corresponding segmentation masks. In some transformations like resize, the same parameters (in this case, scale) should be used for transforming both the images and the segmentation masks. In some transformations, like channel shift, the transformation should not be done on the segmentation masks. Another requirement is that the transformation parameters can differ for each image and its corresponding mask in the batch. Furthermore, a random augmentation should generate different transformation parameters for each image in the batch. Moreover, it should be considered that the transformation does not apply to some images in the batch. Our version of each transformation meets these requirements.

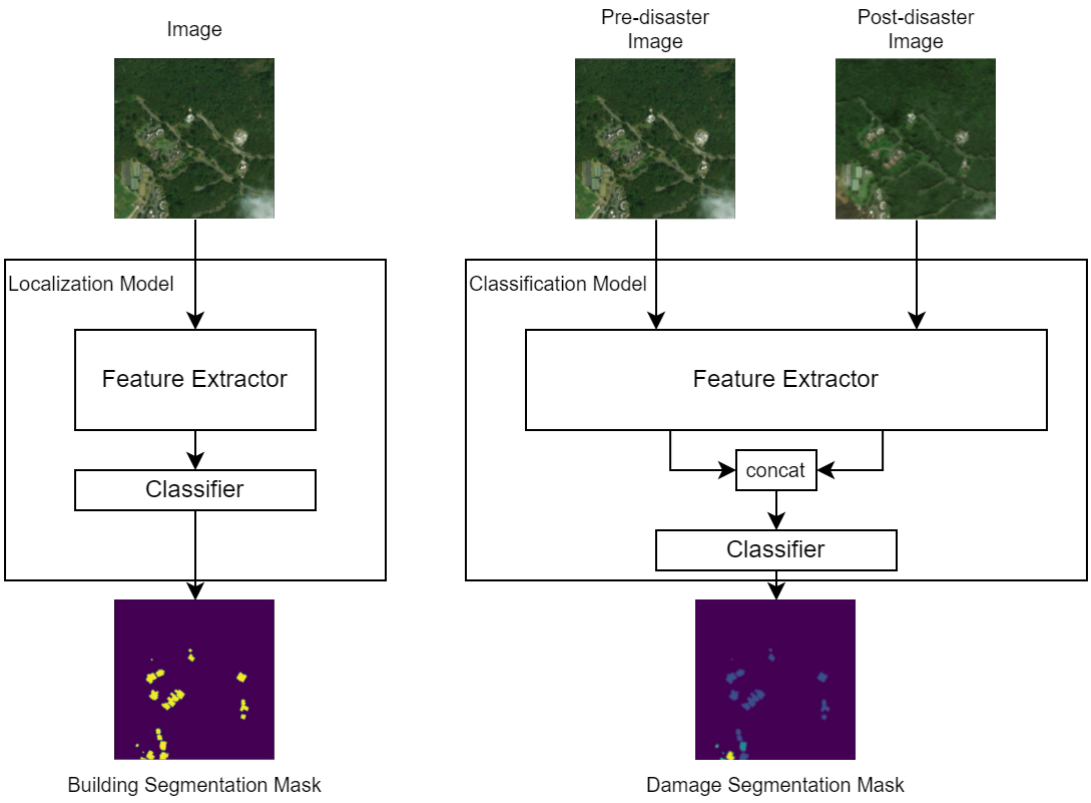


Figure 2: General Model Architecture

## 5 Methodology

### 5.1 General Architecture

As shown in figure 2, building-localization models consist of a feature extractor (a U-net [8] or a SegFormer [9]) and a classifier module [2]. The feature extractor extracts helpful features from the input image; then, the classifier module predicts a value of 0 or 1 for each pixel, indicating whether this pixel belongs to a building or not. The feature extractor module extracts the same features from pre-disaster and post-disaster images in the classification models. In these models, the classifier module predicts a class between 0 and 4 for each pixel. The value 0 indicates that this pixel belongs to no building; values 1-4 mean that this pixel belongs to a building and show the damage level in that pixel. The classifier module learns a distance function between pre-disaster and post-disaster images because the damage level of each facility can be determined by comparing it in the pre- and post-disaster images. In many samples, the post-disaster image has a minor shift compared to the pre-disaster image. However, the segmentation masks are created based on the buildings' location in the pre-disaster image. This shift is an issue the model has to overcome. In our models, feature-extracting weights are shared between the two images. This helps the model to detect the shift or nadir difference. For models that share a joint feature extractor (like *SegFormerB0* Classifier and *SegFormerB0* Localizer), we can initialize the feature extractor module in the classification model with the localization model's feature extractor. Since we do not use the localization model directly for damage assessment, training the localization model can be seen as a pre-training stage for the classification model.

### 5.2 U-Models

Some models in this project use a U-net [8] module as the feature extractor and a superficial 2D Convolutional Layer as the classifier.We call them U-models.

Their feature extractor module is a U-net [8] with five encoder and five decoder modules. Encoder modules are usually a part of a general feature extractor like *Resnet-34* [10]. In the forward pass of each image through each encoder module,

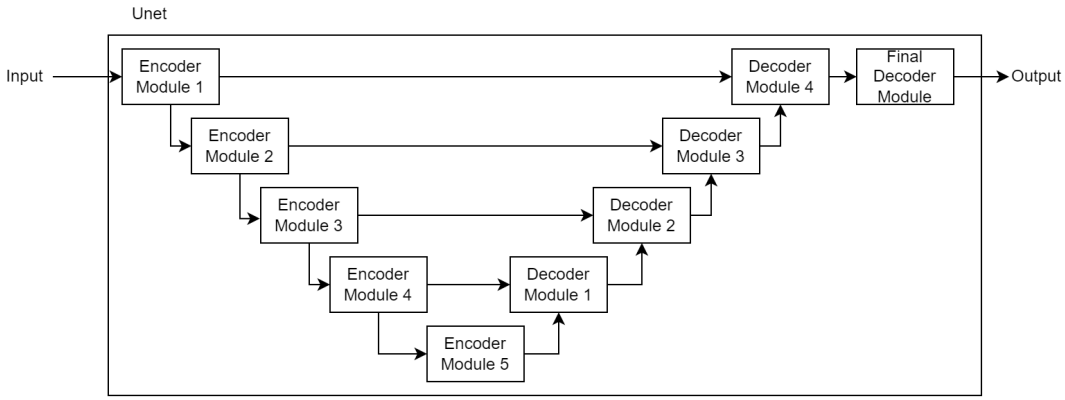


Figure 3: U-models Architecture

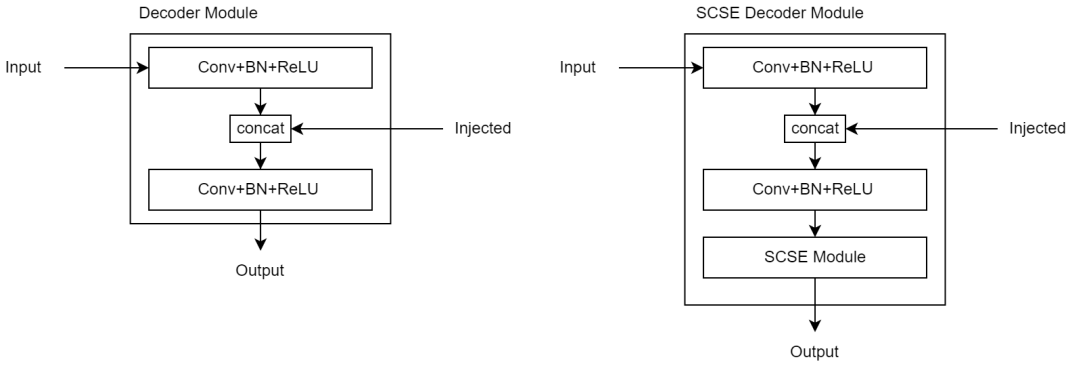


Figure 4: Standard vs SCSE Decoder modules

the number of channels may or may not change. Still, the height and width of the image are divided by two. Usually, the five encoder modules combined include all layers of a general feature extractor model (like Resnet34 [10]) except for the classification layer. Each decoder module combines the output of the previous decoder module and the respective encoder module. For example, encoder module 2 combines the output of decoder module 1 and encoder module 3. They form a U-like structure, as shown in figure 3.

### 5.3 Decoder Modules

There are two variants of decoder modules as shown in figure 4: The *Standard* decoder module and the *SCSE* [11] decoder module.

The *Standard* decoder module applies a 2D convolution and a *ReLU* activation to the input from the previous decoder. Then, it concatenates the result with the input from the respective encoder module and applies another 2D convolution, and *ReLU* activation.

The *SCSE* decoder module works the same way, but in the last step, it uses a *"Concurrent Spatial and Channel Squeeze & Excitation"* [11] module on the result. This SCSE module is supposed to help the model focus on the image's more critical regions and channels. Decoder modules in *Xview2 first place solution* [2] don't use batch normalization between the convolution and the activation. We added this layer to the decoder modules to prevent gradient exploding and to make these modules more stable.

### 5.4 Backbone

We pick encoder modules of U-net modules from a general feature extractor model called *The Backbone Network*. The choice of the backbone network is the most crucial point in the performance of a U-net model. Plus, most of the

Table 1: Models tested and with their attributes

| Model | | Params | Batch Normalization | DecoderType |
|---|---|---|---|---|
| Name | Backbone | | | |
| Resnet34Unet | resnet_34 | 25,728,112 | No | Standard |
| SeResnext50Unet | se_resnext50_32x4d | 34,559,728 | No | Standard |
| Dpn92Unet | dpn_92 | 47,408,735 | No | SCSE - concat |
| SeNet154Unet | senet_154 | 124,874,656 | No | Standard |
| EfficientUnetB0 | efficientnet_b0 | 6,884,876 | Yes | Standard |
| EfficientUnetB0SCSE | | 6,903,860 | Yes | SCSE - no concat |
| EfficientUnetWideSEB0 | efficientnet_widese_b0 | 10,020,176 | Yes | Standard |
| EfficientUnetB4 | efficientnet_b0 | 20,573,144 | Yes | Standard |
| EfficientUnetB4SCSE | | 20,592,128 | Yes | SCSE- no concat |
| SegFormer | segformer_512*512_ade | 3,714,401 | | |

parameters of a U-net model are of its backbone network. Thus, the choice of the backbone network significantly impacts its size and performance. *xview2 first place solution* [2] used *Resnet34* [10], *Dual Path Network 92* [12], *SeResnext50 (32x4d)* [13], and *SeNet154* [14] as the backbone network. We used *EfficientNet B0* and *EfficientNet B4* [15] (both standard and *Wide-SE* versions) as the backbone network, creating new U-models called *Efficient-Unets*. *EfficientNets* [15] have shown excellent results on the *ImageNet* [16] dataset, so they are good feature extractors. They are also relatively small in size. These two features make them perfect choices for a backbone network.

We listed all the used models and their attributes in table 1.

### 5.5 Meta-Learning

In meta-learning, a general problem, such as classifying different images (in the *ImageNet* [16] dataset) or different letters (in the *Omniglot* [17] dataset), is seen as a distribution of tasks. In this approach, tasks are generally the same problem (like classifying letters) but vary in some parameters (like the script letters belong to). We can take a similar approach to our problem. We can view building detection and damage level classification as the general problem and take the disaster type (like a flood, hurricane, or wildfire) and the environment of the disaster (like a desert, forest, or urban area) as the varying factors. In distance-learning methods, the distance function returns a distance between the query sample and each class's sample. Then, the query sample is classified into the class with the minimum distance. These methods are helpful when we have a high number of classes. However, in our case, the number of classes is fixed. Thus, we used a model-agnostic approach. Model agnostic meta-learning [18] algorithms find a set of parameters for the model that can be adapted to a new task by training with very few samples. We used the *MAML* [18] algorithm and considered every different disaster a separate task. Since the *MAML* algorithm consumes lots of memory, and the consumed memory is relative to the model size, we have used models based on EfficientUnetB0 and only trained it for the building localization task.

Since the *MAML* algorithm trains the model much slower than regular training, and we had limited time to train our models, the results weren't satisfactory. We trained *EfficientUnetB0-Localizer* with *MAML* with support shots equal to one or five and query shots equal to two or ten. Other training hyperparameters and evaluation results are available in the results section. We utilized the *Higher* [19] library to implement the *MAML* algorithm.

### 5.6 Vision Transformer

In recent years, vision transformers [20] have achieved state-of-the-art results in many computer vision tasks, including semantic segmentation. SegFormer [9] is a model designed for efficient semantic segmentation, and it is based on vision transformers. SegFormer is available in different sizes. We only used the smallest size, named SegFormerB0. The SegFormer model consists of a hierarchical Transformer encoder and a lightweight all-MLP decode head. In contrast to U-nets, SegFormer models have constant input and output sizes. So, the inputs and outputs of the model should be interpolated to the correct size. For the localization task, the input image goes through SegFormer, and its outputs go through a SegFormer decode head. However, for the classification task, pre and post-disaster go through the

same Segformer model. Next, their outputs are concatenated channel-wise and then go through a modified SegfFormer decode head. The modification is to double the number of channels for the MLP modules. Of course, both outputs can be merged in successive layers, which decreases the distance function complexity. These other versions of the modified decode head can be created and tested in the future. Moreover, one can experiment with changing the size of the SegFormer input and SegFormer model size.

### 5.7 Training Setup

We trained some models with multiple random seeds (multiple folds) to ensure they have low variance and consistent scores. We trained Localization models only on pre-disaster images because post-disaster images added noise to the data; we used post-disaster images in sporadic cases as additional augmentation. We initialized each classification model's feature extractor using weights from the corresponding localization model and fold number. In training both classification and localization models, no weights were frozen. Since the dataset is unbalanced, we use weighted losses with weights relative to the inverse of each class's sample count. We applied morphological dilation with a $5 \times 5$ kernel to classification masks as an augmentation. Dilated masks made predictions bolder. We also used PyTorch [7] amp for FP-16 [21] training.

### 5.8 Loss Functions

Both Building Localization and Damage Classification are semantic segmentation tasks. Because, in both problems, the model's purpose is classification at pixel level. We have used a combination of multiple segmentation losses for all models. In [22], you can find a comprehensive comparison between popular loss functions for semantic segmentation.

Focal and Dice Loss are loss functions used in the training localization models. For Classification models, we tried channel-wise-weighted versions of Focal, Dice, and Cross-entropy Loss.

**Focal Loss** [23]

$$FL(p_t) = -\alpha_t(1 - p_t)\gamma log(p_t). \tag{1}$$

Focal Loss's usage is to make the model focus on hard-to-classify examples by increasing their loss value. We used it because the target distribution was highly skewed. In the building localization task, the number of pixels containing buildings was far less than the background pixels. In the damage classification task, undamaged building samples formed most of the total samples, too.

**Dice Loss** [24]

$$DiceLoss(p, t) = 1 - dice(p, t) \tag{2}$$

Where $dice$, $p$ and $t$ stand for *dice coefficient*, *predictions* and *target values* respectively.

$$dice(A, B) = 2\frac{A \cap B}{A + B} \tag{3}$$

Dice loss is calculated globally over each mini-batch. For multiclass cases, the loss value of each class (channel) is calculated individually, and their average is used as the final loss. Two activation functions can be applied to model outputs before calculating dice loss: *sigmoid* and *softmax*. Softmax makes the denominator of the final loss function constant and thus has less effect on the model's training, though it makes better sense.

**Cross Entropy Loss** [25]

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{4}$$

Since we used sigmoid-dice-loss for multiclass damage classification, cross-entropy loss helped the model assign only one class to each pixel. It solely is a good loss function for semantic segmentation tasks.

# 6 Evaluation

One of the most popular evaluation metrics for classifiers is the F1-score because it accounts for precision and recall simultaneously. The macro version of the F1-score is a good evaluation measure for imbalanced datasets. The Xview2 scoring [26] repository describes what variation of F1-score to use for this problem's scoring. We adapted their evaluation metrics. However, we implemented these metrics as a metric in the Torchmetrics [27] library. It performs better than computing metrics in NumPy [28] and provides an easy-to-use API. The dice score is a set similarity measure that equals the F1-score.

$$Dice(P, Q) = 2.\frac{P \cap Q}{P + Q} \tag{5}$$

$$F1(P, Q) = \frac{2TP}{2TP + FP + FN} \tag{6}$$

## 6.1 Localization Models Scoring

The localization score is defined as a globally calculated binary f1-score. Sample-wise calculation means calculating the score on each sample (image) and then averaging sample scores to get the final score. In global calculation, we use the sum of true positives, true negatives, false positives, and false negatives across all samples to calculate the metric.

The localization score is a binary f1-score, which means class zero (no-building/background) is considered negative, and class one (building) is considered positive. Since we only care about detecting buildings from the background, micro-average is applied too.

## 6.2 Classification Models Scoring

The classification score consists of a weighted sum of 2 scores: the localization score and the damage classification score. Classification models a label of zero to four for each pixel, indicating no-building, no damage, minor damage, major damage, and destroyed, respectively. Since one to four label values show that a specific pixel belongs to a building, we calculate the localization score after converting all values above zero to one. This score determines how good the model is at segmenting buildings. We define the damage classification score as the harmonic mean of the globally computed f1-score for each class from one to four. We calculate the f1-score of each class separately, then use their harmonic mean to give each damage level equal importance. Here, we prefer the harmonic mean to the arithmetic mean because different classes do not have equal support. We compute the damage classification score only on the pixels that have one to four label values in reality. This way, we remove the effect of the models' localization performance from the damage classification score. Hence, these two metrics represent the models' performance in two disparate aspects.

$$score = 0.3 \times F1_{LOC} + 0.7 \times F1_{DC} \tag{7}$$

$$F1_{DC} = 4/(\frac{1}{F1_1 + \epsilon} + \frac{1}{F1_2 + \epsilon} + \frac{1}{F1_3 + \epsilon} + \frac{1}{F1_4 + \epsilon}) \tag{8}$$

## 6.3 Test-Time Augment

While validating a model, we give each piece (or mini-batch) of data to the model and compute a score by comparing the model output and the correct labels. Test-time augment is a technique to enhance the accuracy of the predictions by eliminating the model's bias. For each sample, we use reversible augmentations to generate multiple "transformed samples". The predicted label for the original sample computes as the average of the predicted labels for the " transformed samples". For example, we generate the transformed samples by rotating the original image by 0, 90, 180, and 270 degrees clockwise. Then, we get the model predictions for these transformed samples. Afterward, we rotate the predicted masks 0, 90, 180, and 270 degrees counterclockwise and average them. Their average counts as the model's prediction for the original sample. Using this technique, we eliminate the model's bias of rotation. By reversible augmentation, we mean that no information should be lost during the process of generating "transformed samples" and aggregating their results. For example, in the case of semantic segmentation, shifting an image does not count as a reversible augmentation because it loses some part of the image. However, this technique usually does not

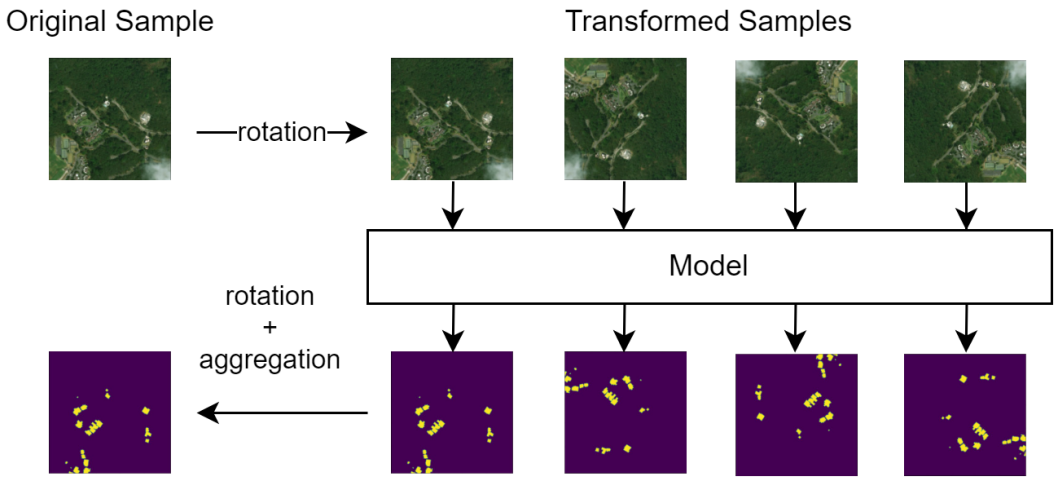


Figure 5: Test-Time Augmentation examples

improve the performance of well-trained models much. Because their bias of a simple thing like rotation is tiny. The same was true for our models when we used flipping and 90-degree rotation as test-time augmentation.

## 7 Experiments

Experiments are done with dataset's *test* directory as the validation set and dataset's *hold* directory as the test set. The results of experiments with localization models are shown in table 2 and the results of experiments with classification models are shown in table 4. Meta-Learning experiments use *mexico-earthquake* and *joplin-tornado* as their test task and their results are shown in table 3.

Using pre-trained feature extractors from localization models allowed classification models to train much faster and have higher scores. Using dilated masks improved accuracy around borders and helped with shifts and different nadirs. The model's classifier module determines each pixel's value based on a distance function between the extracted features from the pre- and post-disaster images. In U-models, the classifier module is a 2D convolution, but in SegFormer models, it is a SegFormer decoder head. Hence, U-models learn a much simpler distance function than SegFormer models; the simplicity of the distance function helps them not to overfit but also prevents them from learning some sophisticated patterns. In the end, SegFormer models train much faster before overfitting on the training data, but U-models slowly reach almost the same score. EfficientUnet localization models have shown that they train better without using focal loss. Softmax dice loss does not perform well in the damage classification model's training. A combination of sigmoid dice loss for each class (channel), and cross-entropy loss gives the best results in the training of a classification model. The effect of SCSE in decoder modules and Wide-SE in Encoder Modules of a U-net is very limited; these variations of EfficientUnets performed almost the same as the standard version.

**Meta-Learning**

## 8 Discussion and Conclusion

Detecting buildings and their damage level by artificial intelligence can improve rescue operations' speed and efficiency after natural disasters. Solving this problem can identify the area of damage on a large scale and prioritize the areas that have been the most affected and damaged by a disaster in rescue operations. We tested different decoders in the U-net modules and utilized different variations of efficient-net as the backbone in our model. Additionally, we fine-tuned SegFormer for our specific task. The result was models with fewer parameters (approximately three million) that performed much better than the previous models (damage classification score=0.77). Due to the fewer parameters, these models have a shorter training and inference time. Therefore, they can be trained and used faster and easily fine-tuned for new and different natural disasters. Considering damage classification and building localization in each natural disaster as a separate task, we utilized *MAML* and trained models that can be adapted to a new natural disaster using

Table 2: The results of experiments with localization models

| Model | Version | Random Seed | Test Score | | Validation Score | |
|---|---|---|---|---|---|---|
| TTA | | | No | Yes | No | Yes |
| Resnet34Unet | 1 | 0 | 0.6590 | 0.6643 | 0.6542 | 0.6590 |
| | | 1 | 0.6690 | 0.6799 | 0.6664 | 0.6768 |
| | | 2 | 0.6839 | 0.6903 | 0.6812 | 0.6858 |
| | | mean agg. | 0.6772 | – | 0.6720 | – |
| SeResnext50Unet | tuned | 0 | 0.6963 | 0.7002 | 0.6957 | 0.6967 |
| | | 1 | 0.7036 | 0.7074 | 0.6916 | 0.6971 |
| | | 2 | 0.7084 | 0.7087 | 0.6981 | 0.7027 |
| | | mean agg. | 0.7088 | – | 0.6998 | – |
| Dpn92Unet | tuned | 0 | 0.6796 | 0.6849 | 0.6776 | 0.6830 |
| | | 1 | 0.6297 | 0.6335 | 0.6335 | 0.6322 |
| | | 2 | 0.6708 | 0.6722 | 0.6662 | 0.6714 |
| | | mean agg. | 0.6597 | – | 0.6637 | – |
| SeNet154Unet | 1 | 0 | 0.7348 | 0.7393 | 0.7261 | 0.7302 |
| | | 1 | 0.7253 | 0.7319 | 0.7100 | 0.7163 |
| | | 2 | 0.7326 | 0.7360 | 0.7217 | 0.7252 |
| | | mean agg. | 0.7409 | – | 0.7264 | – |
| EfficientUnetB0 | Standard | 0 | 0.7692 | 0.7739 | 0.7634 | 0.7666 |
| | | 1 | 0.7685 | 0.7723 | 0.7638 | 0.7662 |
| | | 2 | 0.7704 | 0.7740 | 0.7625 | 0.7666 |
| | SCSE | 0 | 0.7723 | 0.7749 | 0.7644 | 0.7674 |
| | | 1 | 0.7707 | 0.7737 | 0.7628 | 0.7682 |
| | | 2 | 0.7721 | 0.7765 | 0.7647 | 0.7711 |
| | Wide-SE | 0 | 0.7719 | 0.7758 | 0.7662 | 0.7700 |
| | | 1 | 0.7754 | 0.7754 | 0.7664 | 0.7682 |
| EfficientUnetB4 | Standard | 0 | 0.7755 | 0.7797 | 0.7702 | 0.7724 |
| | SCSE | 0 | 0.7811 | 0.7826 | 0.7718 | 0.7743 |
| SegFormerB0 | 512*512_ade | 0 | 0.7602 | 0.7281 | 0.7543 | 0.7214 |
| | | 1 | 0.7569 | 0.7223 | 0.7533 | 0.7189 |
| | | 2 | 0.7605 | 0.7301 | 0.7545 | 0.7250 |

Table 3: The results of experiments with meta-learning models

| model | #tasks | | algorithm | meta-optimizer | | inner-optimizer | | shots | | localization score |
|---|---|---|---|---|---|---|---|---|---|---|
| | train | test | | type | lr | type | lr | support | query | |
| EfficientUnetB0 | 17 | 2 | MAML | AdamW | 15e-6 | SGD | 1e-4 | 1 | 2 | 0.5372 |
| | | | | | 15e-6 | | 1e-3 | 5 | 10 | 0.4351 |

only a few brand-new samples. These models do not have satisfactory performance, but we hope to build better models of this type in the future.

## 9 Future Ideas

The decoder's number of channels can be looked at as a hyper-parameter which can be changed and tuned. Additionally, we can analyze the effect of the size of the backbone in the efficient U-net by trying Efficientnet b5 or b7 as the backbone. The layer in which the embedding of the pre- and post-disaster images get concatenated dictates the complexity of the distance function in the classifier. This effect can also be tested and analyzed. *Log-cosh-dice* and *Focal-Travesky* are two loss functions that have the best performance in the training of segmentation models [22]. We can also try training our models with these two loss functions. But in this case, we have to make sure to modify them, so we can assign weights to classes. The low performance of the meta learning model may not be only due to the small number of training epochs or the small number of shots. We can try using first-order MAML like Reptile [29] instead of the original MAML algorithm in the model. These algorithms use less memory, thus, we can test the effects of other factors and hyperparameters faster. Previous research in the realm of meta-learning for semantic segmentation may also help us train a better model for our specific problem. [30] [31].

Table 4: The results of experiments with classification models

| Model | Version | Random Seed | Test Score | | Validation Score | |
|---|---|---|---|---|---|---|
| TTA | | | NO | Yes | No | Yes |
| Resnet34Unet | tuned | 0 | 0.1090 | 0.0806 | 0.1119 | 0.0831 |
| | | 1 | 0.1466 | 0.1174 | 0.1264 | 0.0997 |
| | | 2 | 0.1314 | 0.1101 | 0.1324 | 0.1082 |
| | | mean agg. | 0.0860 | – | 0.0832 | – |
| SeResnext50Unet | tuned | 0 | 0.6164 | 0.6152 | 0.6397 | 0.6347 |
| | | 1 | 0.6135 | 0.6069 | 0.6012 | 0.5991 |
| | | 2 | 0.6319 | 0.6422 | 0.6271 | 0.6361 |
| | | mean agg. | 0.6360 | – | 0.6301 | – |
| Dpn92Unet | tuned | 0 | 0.6564 | 0.6657 | 0.6387 | 0.6441 |
| | | 1 | 0.6233 | 0.6343 | 0.5869 | 0.5813 |
| | | 2 | 0.6246 | 0.6252 | 0.6075 | 0.6138 |
| | | mean agg. | 0.6460 | – | 0.6258 | – |
| SeNet154Unet | tuned | 0 | 0.6916 | 0.7034 | 0.6684 | 0.6722 |
| | | 1 | 0.6216 | 0.6342 | 0.5889 | 0.6123 |
| | | 2 | 0.6868 | 0.6949 | 0.6520 | 0.6479 |
| | | mean agg. | 0.6954 | – | 0.6596 | – |
| EfficientNetB0 | Standard | 0 | 0.7576 | 0.7571 | 0.7606 | 0.7505 |
| | SCSE | 0 | 0.7591 | 0.7525 | 0.7497 | 0.7399 |
| | Wide-SE | 0 | 0.7726 | 0.7667 | 0.7769 | 0.7737 |
| EfficientUnetB4 | Standard | 0 | 0.7732 | 0.7679 | 0.7684 | 0.7589 |
| | SCSE | 0 | 0.7746 | 0.7650 | 0.7740 | 0.7635 |
| SegFormer | Standard | 0 | 0.7574 | 0.7380 | 0.7385 | 0.6993 |

## References


[1] Ritwik Gupta, Richard Hosfelt, Sandra Sajeev, Nirav Patel, Bryce Goodman, Jigar Doshi, Eric Heim, Howie Choset, and Matthew Gaston. xbd: A dataset for assessing building damage from satellite imagery, 2019.

[2] vdurnov. 1st place solution for "xview2: Assess building damage" challenge. `https://github.com/vdurnov/xview2_1st_place_solution`, 2020.

[3] Xview2. Computer vision for building damage assessment using satellite imagery of natural disasters, 2020.

[4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. Kornia: an open source differentiable computer vision library for pytorch, 2019.

[6] E. Riba, D. Mishkin, J. Shi, D. Ponsa, F. Moreno-Noguer, and G. Bradski. A survey on kornia: an open source differentiable computer vision library for pytorch, 2020.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[9] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers, 2021.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[11] Abhijit Guha Roy, Nassir Navab, and Christian Wachinger. Concurrent spatial and channel squeeze & excitation in fully convolutional networks, 2018.

[12] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks, 2017.

[13] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2017.

[14] Abhijit Guha Roy, Nassir Navab, and Christian Wachinger. Recalibrating fully convolutional networks with spatial and channel 'squeeze & excitation' blocks, 2018.

[15] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[17] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. The omniglot challenge: a 3-year progress report, 2019.

[18] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

[19] Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. Generalized inner loop meta-learning, 2019.

[20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[21] Pradeep V Kotipalli, Ranvijay Singh, Paul Wood, Ignacio Laguna, and Saurabh Bagchi. Ampt-ga: automatic mixed precision floating point tuning for gpu applications. In *Proceedings of the ACM International Conference on Supercomputing*, ICS '19, page 160–170, New York, NY, USA, 2019. Association for Computing Machinery.

[22] Shruti Jadon. A survey of loss functions for semantic segmentation. In *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. IEEE, October 2020.

[23] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018.

[24] Carole H. Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M. Jorge Cardoso. *Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations*, page 240–248. Springer International Publishing, 2017.

[25] Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels, 2018.

[26] CIUx-xView. Reference code showing how scores/metrics are computed for the xview2 challenge. `https://github.com/DIUx-xView/xView2_scoring`, 2019.

[27] Nicki Skafte Detlefsen, Jiri Borovec, Justus Schock, Ananya Harsh, Teddy Koker, Luca Di Liello, Daniel Stancl, Changsheng Quan, Maxim Grechkin, and William Falcon. TorchMetrics - Measuring Reproducibility in PyTorch, February 2022.

[28] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[29] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms, 2018.

[30] Shuai Luo, Yujie Li, Pengxiang Gao, Yichuan Wang, and Seiichi Serikawa. Meta-seg: A survey of meta-learning for image segmentation. *Pattern Recognition*, 126:108586, 2022.

[31] Sean M. Hendryx, Andrew B. Leach, Paul D. Hein, and Clayton T. Morrison. Meta-learning initializations for image segmentation, 2020.